Relazione

HOTELIER: an HOTEL advIsor sERvice

Progetto di Reti A.A. 2023/2024

Lorenzo Pernigoni 597958

Indice

1 Albero della directory

```
.
├── client.jar
├── server.jar
├── client.properties
├── server.properties
├── bin
│   └── ...tutti i file .class
├── json
│   ├── Cities.json
│   ├── Hotels.json
│   ├── Reviews.json
│   └── Users.json
├── lib
│   └── gson-2.10.1.jar
└── src
    ├── client
    │   ├── ClientMain.java
    │   └── LocalRankingReceiver.java
    ├── entities
    │   ├── Hotel.java
    │   ├── Ratings.java
    │   ├── Review.java
    │   ├── StatusClient.java
    │   ├── StatusRegistration.java
    │   └── User.java
    ├── rmi
    │   ├── NotifyClientImpl.java
    │   ├── NotifyClientInterface.java
    │   ├── NotifyServerImpl.java
    │   ├── NotifyServerInterface.java
    │   ├── UserHashMap.java
    │   └── UserHashMapInterface.java
    ├── server
    │   ├── DataPersistenceTask.java
    │   ├── LocalRankingUpdater.java
    │   ├── ServerMain.java
    │   ├── TerminationHandler.java
    │   └── Worker.java
    └── utils
        └── PasswordUtils.java
```

2 Json

Cities.json File che contiene le città capoluogo di regione e di provincia italiane.

Hotels.json File utilizzato per memorizzare in modo permanente i dati degli hotel. Non devono essere inseriti più hotel con lo stesso nome nella stessa città.

Reviews.json File utilizzato per memorizzare in modo permanente i dati delle recensioni degli hotel fatte dagli utenti registrati.

Users.json File utilizzato per memorizzare in modo permanente i dati degli utenti registrati.

3 Entities

Hotel.java Classe che rappresenta un hotel. Notare, tra gli attributi, rate (di tipo double, punteggio sintetico calcolato in base alle recensioni dell'hotel) e ratings (di tipo Ratings, punteggi delle categorie calcolati in base alle recensioni dell'hotel).

User.java Classe che rappresenta un utente registrato. Notare, tra gli attributi, experienceLevel (livello di esperienza dell'utente, da 0 a 5) e numReviews (numero di recensioni inserite dall'utente), entrambi inizialmente impostati a 0. Il metodo pubblico `incrNumReviews()` permette di incrementare di uno il numero di recensioni inserite dall'utente e, in base ad esso, aggiorna il suo livello di esperienza.

Review.java Classe che rappresenta una recensione di un hotel. Attributi: hotelName (nome dell'hotel recensito), city (città in cui si trova l'hotel recensito), reviewer (username dell'utente che ha fatto la recensione), rate (di tipo int, punteggio sintetico, da 0 a 5), ratings (di tipo Ratings, punteggi delle categorie, da 0 a 5), dateTime (data e ora in cui è stata inserita la recensione).

Ratings.java Classe che rappresenta i punteggi relativi alle categorie cleaning, position, services, quality.

StatusRegistration.java Classe enum che rappresenta i possibili valori di ritorno per il metodo remoto `register()`. Può assumere uno dei seguenti valori: `SUCCESS`, `USERNAME_TAKEN`, `BLANK_PASSWORD`, `BLANK_USERNAME`, `TOO_LONG`.

StatusClient.java Classe enum che rappresenta lo stato dell'utente su un client. Può assumere uno dei seguenti valori: `USER_NOT_LOGGED` (nessun utente loggato sul client), `USER_LOGGED` (utente loggato sul client), `EXIT` (l'utente, loggato o non loggato, vuole uscire dal sistema).

4 Server

4.1 Strutture dati

hotelsByCityMap ConcurrentHashMap che ha come chiave il nome di una città (String) e come valore la lista degli hotel presenti in quella città (CopyOnWriteArrayList<Hotel>).

reviewsMap ConcurrentHashMap che ha come chiave il nome dell'hotel seguito da un underscore seguito dal nome della città in cui si trova l'hotel (String, e.g. `"Hotel Firenze 1_Firenze"`) e come valore la lista di recensioni di quell'hotel (CopyOnWriteArrayList<Review>).

usersMap ConcurrentHashMap che ha come chiave l'username di un utente registrato (String) e come valore l'utente corrispondente (User).

4.2 ServerMain.java

Classe che contiene il metodo `main()` del server.

- Crea hotelsByCityMap e reviewsMap.

- Legge il file di configurazione server.properties con il metodo `readConfig()`. Notare, in particolare, i campi: persistencePeriod (periodo di tempo tra un salvataggio delle strutture dati in json e l'altro, in secondi), rankingPeriod (periodo di tempo tra un ricalcolo della classifica locale e l'altro, in secondi), sameReviewerSameHotelPeriod (periodo di tempo che deve trascorrere tra le recensioni dello stesso utente per lo stesso hotel, in secondi).

- Chiama i metodi `loadCitiesFromJson()`, `loadHotelsFromJson()`, `loadReviewsFromJson()` che utilizzano Gson Streaming API.

  `loadCitiesFromJson()` legge le città dal file Cities.json e le inserisce come chiave in hotelsByCityMap. Come valore crea un CopyOnWriteArrayList vuoto per ogni chiave.

  `loadHotelsFromJson()` legge gli hotel dal file Hotels.json, li deserializza e li aggiunge alla lista (valore di hotelsByCityMap) che ha come chiave la città in cui esso si trova.

  `loadReviewsFromJson()` legge le recensioni dal file Reviews.json, le deserializza e le aggiunge alla lista (valore di reviewsMap) con chiave "nomeHotel_città" dell'hotel di cui è stata fatta la recensione.

  Le liste presenti in hotelsByCityMap vengono ordinate in modo decrescente in base al rate (punteggio sintetico).

- **RMI per la registrazione di un utente** (vedi anche 6.1)

  Crea l'oggetto users di tipo UserHashMap. In usersMap, prende un riferimento alla ConcurrentHashMap dell'oggetto users invocando il metodo `getUsersMap()`.

  Chiama il metodo `loadUsersFromJson()` per leggere gli utenti già registrati dal file Users.json, deserializzarli uno ad uno, ed inserirli in usersMap (con chiave username e valore l'oggetto User deserializzato).

  Esporta users ottenendo lo stub corrispondente e crea un registry RMI in cui pubblica lo stub.

- Per la **persistenza delle strutture dati sui file json**, utilizza uno ScheduledExecutorService single-threaded che esegue il task **DataPersistenceTask** ogni 'persistencePeriod' secondi.

  Viene registrato anche uno shutdown hook che, alla terminazione del server, permetterà di persistere i dati un'ultima volta.

- **RMI per il servizio di notifica** (vedi anche 6.2)

  Crea l'oggetto server di tipo NotifyServerImpl, lo esporta ottenendo lo stub corrispondente e pubblica lo stub nel registry RMI.

- Per **ricalcolare periodicamente le classifiche locali e inviare le notifiche**, utilizza uno ScheduledExecutorService single-threaded che esegue il task **LocalRankingUpdater** ogni 'rankingPeriod' secondi.

- **Interagisce con il client** secondo il modello richiesta/risposta su una connessione TCP. Per fare ciò crea un server socket su cui accettare le connessioni dei client e gestisce ogni connessione grazie a un **Worker** eseguito in un cached thread pool.

  Viene registrato anche uno shutdown hook che, alla terminazione del server, chiuderà ordinatamente il thread pool e il server socket eseguendo il thread **TerminationHandler**.

4.3 DataPersistenceTask.java

Classe che rappresenta il task da eseguire per gestire il salvataggio dei dati in formato json.

Dentro al metodo `run()` chiama i metodi `persistUsers()`, `persistReviews()`, `persistHotels()` che utilizzano Gson Streaming API per scrivere token per token dentro ai file json.

- `persistUsers()` salva i dati degli utenti presenti in usersMap nel file Users.json.
- `persistReviews()` salva i dati delle recensioni presenti in reviewsMap nel file Reviews.json.
- `persistHotels()` salva i dati degli hotel presenti in hotelsByCityMap nel file Hotels.json.

4.4 LocalRankingUpdater.java

Classe che rappresenta il task che: (1) ricalcola e aggiorna rate (punteggio sintetico) e ratings (punteggi delle categorie) degli hotel; (2) se è cambiato il primo in classifica di una classifica locale, invia una notifica ai client iscritti al gruppo di multicast; (3) se almeno uno tra i primi 3 di una classifica locale è cambiato, notifica la variazione della classifica con una callback RMI a tutti i client registrati.

4.4.1 Aggiornamento dei punteggi e descrizione dell'Algoritmo

Per ogni lista di recensioni presente in reviewsMap (quindi per ogni hotel): (1) calcola, usando il metodo statico `calculateTotalScore()`, il rate dell'hotel basato sui rate delle sue recensioni; (2) calcola il ratings dell'hotel come media dei ratings delle sue recensioni; (3) setta, in hotelsByCityMap, i nuovi valori di rate e ratings appena ricalcolati relativi all'hotel in questione.

Algoritmo L'algoritmo per il calcolo del rate di un hotel è rappresentato dal metodo statico `calculateTotalScore()` che prende come argomento una lista di recensioni. Deve tenere in considerazione la qualità, la quantità e l'attualità di ogni recensione.

- La **qualità** è la media dei rate delle recensioni e viene calcolata dal metodo `calculateAverageQuality()`.
- La **quantità** è il numero totale di recensioni della lista.
- L'**attualità** tiene conto della distanza temporale tra le date delle recensioni e la data attuale, con un peso maggiore per le recensioni più recenti.

  I pesi di attualità di ogni recensione vengono calcolati dal metodo `calculateRecencyFactor()` e vengono poi usati per calcolare una media pesata dei rate.
- Il rate **totale** è una combinazione di qualità (peso 0.4), quantità (peso 0.2) e attualità (peso 0.4).

4.4.2 Notifiche Multicast UDP

Per ogni lista di hotel presente in hotelsByCityMap (quindi per ogni città), se il primo in classifica è cambiato, invia ai client iscritti al gruppo di multicast un pacchetto UDP (DatagramPacket) contenente un array di byte che rappresenta una notifica (String). Esempio di notifica: `"[NOTIFICA] Il nuovo primo in classifica a Milano è Hotel Milano 7"`.

Come capire se il primo in classifica è cambiato? All'inizio del `run()` viene fatta una deep copy (in hotelsByCityMap_Old) delle prime 3 posizioni di ogni città di hotelsByCityMap (ovvero dei primi 3 elementi delle liste di hotel, dato che sono ordinate in base al rate). Dopo l'aggiornamento dei punteggi, si riordinano le classifiche locali in hotelsByCityMap e si confrontano gli id del nuovo e del vecchio hotel primo in classifica.

4.4.3 Notifiche RMI

Per ogni lista di hotel presente in hotelsByCityMap (quindi per ogni città), se almeno una tra le prime 3 posizioni della classifica è cambiata, notifica la variazione della classifica con una callback per ogni client registrato: `server.update(city, hotelNames)`.

Come capire se almeno una tra le prime 3 posizioni della classifica è cambiata? Nello stesso modo in cui si capisce se il primo in classifica è cambiato, come spiegato precedentemente.

4.5 Worker.java

Classe che rappresenta il task che si occupa di interagire con un client su una connessione TCP. Il worker riceve un comando dal client, esegue l'azione richiesta e invia la risposta.

Risponde al client con una stringa con il formato: `[stato],[contenuto]\n`
`[stato]` indica lo stato dell'utente sul client (StatusClient), mentre `[contenuto]` indica il testo del messaggio di risposta.

Utilizza la variabile **status**, di tipo StatusClient, per tenere traccia dello stato dell'utente. Essa viene inizializzata a `USER_NOT_LOGGED`, settata a `USER_LOGGED` in caso di login, settata a `USER_NOT_LOGGED` in caso di logout, settata a `EXIT` in caso di exit. Inoltre, se l'utente è loggato, viene memorizzato il suo username nella variabile **usernameLogged**.

Di seguito vengono descritti i comandi gestiti dal worker e utilizzati dal client.

- `help`
  Invia un messaggio di aiuto al client in cui elenca i comandi supportati dal sistema.

- `exit`
  Se l'utente è loggato fa in automatico il logout.
  In ogni caso setta status a `EXIT`, invia la risposta ed esce dal ciclo `while`.

- `login <username> <password>`
  Chiama il metodo `login()` che tenta di effettuare il login dell'utente e invia al client un messaggio contenente l'esito dell'operazione. Se il login ha avuto successo, la risposta del worker conterrà anche la richiesta di inviare le città che il client vuole seguire per riceverne gli aggiornamenti sulla classifica. Le città sono richieste tutte su una riga, con uno spazio tra una e l'altra. Il worker controlla l'esistenza delle città inserite dal client e invia la risposta con quelle giuste.

- `logout <username>`
  Chiama il metodo `logout()` che tenta di effettuare il logout dell'utente e, se il logout è stato espressamente richiesto dal client, gli invia un messaggio contenente l'esito dell'operazione.

- `searchHotel <nomeHotel> <città>`
  Chiama il metodo `searchHotel()` che cerca e invia al client i dati dell'hotel richiesto o un messaggio di errore.
  Esempio di comando scritto correttamente: `searchHotel Hotel Milano 7 Milano`

- `searchAllHotels <città>`
  Chiama il metodo `searchAllHotels()` che cerca e invia al client i dati di tutti gli hotel presenti nella città richiesta o un messaggio di errore.
  Esempio di comando scritto correttamente: `searchAllHotels Milano`

- `insertReview`
  `<nomeHotel> <città> <rate> <cleaning> <position> <services> <quality>`

  Chiama il metodo `insertReview()` che tenta di inserire la recensione di un hotel che l'utente loggato ha chiesto di inserire e gli comunica l'esito dell'operazione.
  Esempio di comando scritto correttamente:
  `insertReview Hotel Milano 7 Milano 4 5 4 4 3`

- `showMyBadges`
  Chiama il metodo `insertReview()` che mostra all'utente loggato il distintivo corrispondente al suo livello di esperienza. Il distintivo è una stringa.

4.6 TerminationHandler.java

Classe che rappresenta il thread che gestisce la terminazione controllata del server.

- Chiude il server socket in modo da sbloccare il server bloccato sulla `accept()` e non accettare più nuove connessioni.
- Avvia la terminazione del thread pool. Se esso non termina entro 'maxDelay' millisecondi, lo fa terminare forzatamente.

5 Client

5.1 Strutture dati

notifyRankedFirstQueue Coda (LinkedBlockingQueue<String>) per gestire in modo asincrono le notifiche riguardanti il cambiamento della prima posizione delle classifiche locali.

updatedRankingMap ConcurrentHashMap che contiene gli aggiornamenti delle classifiche locali. Ha come chiave la città della classifica locale (String) e come valore la lista che contiene i nomi di, al massimo, i primi 3 hotel della nuova classifica in ordine decrescente di rate (CopyOnWriteArrayList<String>).

followedCities Lista (CopyOnWriteArrayList<String>) che contiene i nomi delle città di cui l'utente, immediatamente dopo il login, ha espresso l'interesse di seguirne la classifica.

5.2 ClientMain.java

Classe che contiene il metodo `main()` del client.

- Legge il file di configurazione client.properties con il metodo `readConfig()`.
- Crea un socket (per connettersi con il server) e i relativi stream di input/output.
- **RMI per la registrazione di un utente** (vedi anche 6.1)

  Prende il riferimento (in u) a un oggetto remoto UserHashMapInterface dal registry.
- **RMI per il servizio di notifica** (vedi anche 6.2)

  Prende il riferimento (in server) a un oggetto remoto NotifyServerInterface dal registry.

  Crea l'oggetto remoto callbackObj, di tipo NotifyClientInterface, e lo esporta ottenendo lo stub corrispondente.

- Utilizza la variabile **status**, di tipo StatusClient, per tenere traccia dello stato dell'utente. È inizializzata a `USER_NOT_LOGGED` e viene aggiornata in base alle risposte ricevute dal worker.

- **Ciclo while(true)**

  - All'inizio, se l'utente è loggato, prende e rimuove ogni elemento da notifyRankedFirstQueue e lo stampa (sono le notifiche arrivate dal gruppo di multicast).
  - Legge l'input dell'utente dalla CLI con uno Scanner.
  - Interpreta i comandi inseriti dall'utente.
    - `register <username> <password>`
      Usa RMI per per registrare l'utente non loggato invocando il metodo remoto `u.register(username, password)` e, a seconda del risultato, stampa l'esito dell'operazione.
    - `myRankings`
      Permette all'utente loggato di vedere gli aggiornamenti delle prime 3 posizioni delle classifiche delle città a cui ha espresso il proprio interesse dopo il login. Per stampare queste informazioni usa le strutture dati updatedRankingMap e followedCities.
    - `help | exit | login | logout | searchHotel | searchAllHotels | insertReview | showMyBadges`
      Questi comandi sono gestiti dal server, in particolare dal thread worker (il formato dei messaggi di risposta è quello descritto precedentemente, vedi 4.5).
      Il client invia quindi l'input dell'utente al server, legge la risposta, aggiorna la variabile status e stampa il contenuto del messaggio ricevuto.
      Subito dopo aver effettuato il **login**, come richiesto dal server l'utente invia le città che vuole seguire per riceverne gli aggiornamenti sulla classifica. A questo punto il client si registra al servizio di notifica RMI con `server.registerForCallback(stub)` e, inoltre, avvia il thread LocalRankingReceiver che si iscrive al gruppo di multicast per ricevere notifiche riguardanti il cambiamento della prima posizione delle classifiche locali.
      Subito dopo aver effettuato il **logout**, invece, cancella la registrazione al servizio di notifica RMI con `server.unregisterForCallback(stub)`, arresta il thread LocalRankingReceiver e fa una `clear()` delle strutture dati usate per gestire le notifiche.

5.3 LocalRankingReceiver.java

Classe che rappresenta il thread che gestisce la ricezione, da un multicast socket, delle notifiche riguardanti il cambiamento della prima posizione delle classifiche locali. Ogni pacchetto UDP ricevuto contiene un array di byte che viene decodificato in una stringa da aggiungere alla coda notifyQueue.

Viene implementato anche il metodo pubblico `shutdown()` che sarà chiamato dal `main()` del client per arrestare questo thread.

6 RMI

6.1 Registrazione di un utente

UserHashMapInterface.java
Interfaccia che estende Remote e definisce il metodo `register()`.

UserHashMap.java Classe che implementa UserHashMapInterface.

- Ha come attributo usersMap, ovvero la ConcurrentHashMap utilizzata per memorizzare gli utenti registrati.
- Implementa il metodo remoto `register()`. Questo metodo, che sarà esportato dal server e invocato dal client, serve per registrare un nuovo utente inserendolo in usersMap. Prende come argomenti l'username e la password dell'utente che intende registrarsi e restituisce un oggetto di tipo StatusRegistration per informare il client dell'esito dell'operazione. In caso di successo i dati dell'utente vengono inseriti in usersMap. La password non viene memorizzata in chiaro ma viene hashata usando i metodi `generateSalt()` e `hashPassword()` della classe **PasswordUtils.java**.
- Implementa il metodo `getUsersMap()`, non esposto nell'interfaccia, che serve al server per ottenere un riferimento a usersMap.

6.2 Notifiche per gli aggiornamenti delle classifiche locali

Servizio di notifica per gli aggiornamenti delle classifiche locali (solo delle prime 3 posizioni), implementato con il meccanismo di callback RMI.

NotifyClientInterface.java
Interfaccia che estende Remote e definisce il metodo `notifyUpdatedRanking()`.

NotifyClientImpl.java Classe che implementa NotifyClientInterface.

- Ha come attributo **updatedRankingMap**, ovvero la ConcurrentHashMap utilizzata per contenere gli aggiornamenti delle classifiche locali. Essa ha come chiave la città della classifica locale (String) e come valore la lista che contiene i nomi di, al massimo, i primi 3 hotel della nuova classifica in ordine decrescente di rate (CopyOnWriteArrayList<String>).
- Il metodo costruttore `NotifyClientImpl()` assegna la ConcurrentHashMap passata come argomento a updatedRankingMap. La funzionalità del costruttore è quella di creare un oggetto remoto callback client.
- Implementa il metodo remoto `notifyUpdatedRanking()` che inserisce in updatedRankingMap la classifica locale aggiornata passata come argomento.

NotifyServerInterface.java
Interfaccia che estende Remote e definisce i metodi `registerForCallback()` e `unregisterForCallback()`.

NotifyServerImpl.java Classe che implementa NotifyServerInterface.

- Ha come attributo una lista clients (List<NotifyClientInterface>) che conterrà i client registrati, ovvero gli stub.
- Per creare un oggetto remoto server, verrà utilizzato il metodo costruttore `NotifyServerImpl()`.
- Implementa il metodo remoto `registerForCallback()` per la registrazione della callback (la callback è lo stub del client, oggetto che consente al server di invocare un metodo sul client). Questo metodo, esportato dal server e invocato dal client, aggiunge la stub del client passata come argomento alla lista clients.
- Implementa il metodo remoto `unregisterForCallback()` per la cancellazione della registrazione per la callback. Questo metodo, esportato dal server e invocato dal client, rimuove la stub del client passata come argomento dalla lista clients.
- Implementa il metodo `update()` che al suo interno invoca il metodo `doCallbacks()`. `update()` viene invocato dal server, in particolare dal thread LocalRankingUpdater (vedi 4.4.3).
- Implementa il metodo `doCallbacks()` che ha il compito di eseguire le callback invocando il metodo remoto del client `notifyUpdatedRanking()` su ogni elemento della lista clients.

7 Istruzioni per compilazione ed esecuzione

Il codice del progetto è stato scritto utilizzando la versione Java 22.

Compilazione

```
javac -d bin -sourcepath src -cp lib/gson-2.10.1.jar src/client/*.java
src/server/*.java src/entities/*.java src/rmi/*.java src/utils/*.java
```

Esecuzione su Windows

Server: `java -cp "lib/gson-2.10.1.jar;bin" server.ServerMain`
Client: `java -cp bin client.ClientMain`

Esecuzione su Linux

Server: `java -cp "lib/gson-2.10.1.jar:bin" server.ServerMain`
Client: `java -cp bin client.ClientMain`

Esecuzione usando i file JAR

Server: `java -jar server.jar`
Client: `java -jar client.jar`

8 Esempio di utilizzo di un client

```
PS C:\Users\User\Progetto> java -cp bin client.ClientMain
> login lorenzo 1234
Login avvenuto con successo
Inserisci le città di cui vuoi seguire la classifica (tutte nella prossima riga, ...
> Torino Napoli Parigi Londra
Città seguite: Torino Napoli
> myRankings
Nessun aggiornamento dal momento del tuo login
> insertReview Hotel Torino 9 Torino 2 2 3 1 2
Recensione inserita correttamente
> myRankings
Torino
  (1) Hotel Torino 7
  (2) Hotel Torino 3
  (3) Hotel Torino 9
> insertReview Hotel Napoli 3 Napoli 4 3 4 3 4
Recensione inserita correttamente
>
>
[NOTIFICA] Il nuovo primo in classifica a Napoli è: Hotel Napoli 3
>
> insertReview Hotel Napoli 6 Napoli 3 3 4 3 4
Recensione inserita correttamente
> myRankings
Napoli
  (1) Hotel Napoli 3
  (2) Hotel Napoli 7
  (3) Hotel Napoli 6
Torino
  (1) Hotel Torino 7
  (2) Hotel Torino 3
  (3) Hotel Torino 9
> showMyBadges
--- Contributore Super ---
> logout lorenzo
Hai effettuato il logout
>
> login vale_ 5678
Login avvenuto con successo
Inserisci le città di cui vuoi seguire la classifica (tutte nella prossima riga, ...
>
Città seguite: nessuna
> showMyBadges
--- Recensore Esperto ---
> logout vale_
Hai effettuato il logout
>
> searchHotel Hotel Firenze 7 Firenze
```

```
Hotel Firenze 7
  "Un organizzato hotel a Firenze, in Via Roma, 62"
  phone=391-4208074
  services=[Servizio in camera, Wi-Fi]
  rate=4.17
  ratings=[5.00, 5.00, 5.00, 5.00]

> register franco 8888
Registrazione avvenuta con successo
> login franco 8888
Login avvenuto con successo
Inserisci le città di cui vuoi seguire la classifica (tutte nella prossima riga, ...
> Pisa
Città seguite: Pisa
> exit
Logout automatico
Esco dal client
[CLIENT] Terminato
PS C:\Users\User\Progetto>
```